**SCALA MINI-PROJECT**

**Aim –**

Write a simple program in the SCALA using apache spark framework.

**Objective –**

**Theory –**

**1. What is Scala –**

Scala is an acronym for “Scalable Language”. It is a general-purpose programming language designed for the programmers who want to write programs in a concise, elegant, and type-safe way. Scala enables programmers to be more productive. Scala is developed as an object-oriented and functional programming language.

**2. About Scala**

The design of Scala started in 2001 in the programming methods laboratory at EPFL (École Polytechnique Fédérale de Lausanne). Scala made its first public appearance in January 2004 on the JVM platform and a few months later in June 2004, it was released on the .(dot)NET platform. The .(dot)NET support of Scala was officially dropped in 2012. A few more characteristics of Scala are:

2.1 Scala is pure Object-Oriented programming language
2.2 Scala is a functional language

The functional programming exhibits following characteristics:

- Power and flexibility
- Simplicity
- Suitable for parallel processing

2.3 Scala is a compiler based language (and not interpreted)
2.4 Companies using Scala
Scala is now big name. It is used by many companies to develop the commercial software. These are the following notable big companies which are using Scala as a programming alternative.

- LinkedIn
- Twitter
- Netflix
- Apple

**3. Installing Scala**

**Step 0:** Open the terminal

**Step 1:** Install Java

```
$ sudo apt-add-repository ppa:webupd8team/java
```

```
$ sudo apt-get update
$ sudo apt-get install oracle-java7-installer
```

If you are asked to accept Java license terms, click on “Yes” and proceed. Once finished, let us check whether Java has installed successfully or not. To check the Java version and installation, you can type:

```
$ java -version
```

**Step 2:** Once Java is installed, we need to install Scala

```
$ cd ~/Downloads
$ wget http://www.scala-lang.org/files/archive/scala-2.11.7.deb
$ sudo dpkg -i scala-2.11.7.deb
$ scala –version
```

This will show you the version of Scala installed

**4. Prerequisites for Learning Scala**

Scala being an easy to learn language has minimal prerequisites. If you are someone with basic knowledge of C/C++, then you will be easily able to get started with Scala.

**5. Choosing a development environment**

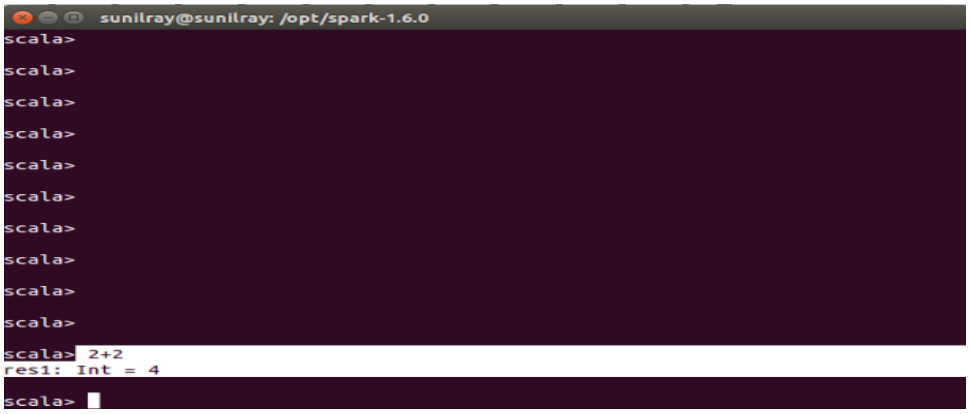
Once you have installed Scala, there are various options for choosing an environment. Here are the 3 most common options:

- Terminal / Shell based
- Notepad / Editor based
- IDE (Integrated development environment)

Choosing right environment depends on your preference and use case.

**Warming up: Running your first Scala program in Shell:**

Let’s write a first program which adds two numbers.




**6. Scala Basics Terms**

**Object**: An entity that has state and behavior is known as an object. For example: table, person, car etc.

**Class:** A class can be defined as a blueprint or a template for creating different objects which defines its properties and behavior.

**Method:** It is a behavior of a class. A class can contain one or more than one method. For example: deposit can be considered a method of bank class.

**Closure:** Closure is any function that closes over the environment in which it's defined. A closure returns value depends on the value of one or more variables which is declared outside this closure.

**Traits:** Traits are used to define object types by specifying the signature of the supported methods. It is like interface in java.

**7. Things to note about Scala**

- It is case sensitive
- If you are writing a program in Scala, you should save this program using ".scala"
- Scala execution starts from main() methods
- Any identifier name cannot begin with numbers. For example, variable name "123salary" is invalid.
- You can not use Scala reserved keywords for variable declarations or constant or any identifiers.

**8. Variable declaration in Scala**

In Scala, you can declare a variable using ‘var’ or ‘val’ keyword. The decision is based on whether it is a constant or a variable. If you use ‘var’ keyword, you define a variable as mutable variable. On the other hand, if you use ‘val’, you define it as immutable. Let’s first declare a variable using “var” and then using “val”.

8.1 Declare using var

```
var Var1 : String = "Ankit"
```

8.2 Declare using val

```
val Var2 : String = "Ankit"
```

**9. Operations on variables**

You can perform various operations on variables. There are various kinds of operators defined in Scala.

For example: Arithmetic Operators, Relational Operators, Logical Operators, Bitwise Operators, Assignment Operators.

```
scala> var Var4 = 2
Output: Var4: Int = 2
scala> var Var5 = 3
Output: Var5: Int = 3
```

Now, let us apply some operations using operators in Scala.

Apply ‘+’ operator

```
Var4+Var5
Output:
res1: Int = 5
```

**Apply “==” operator**

```
Var4==Var5
Output:
res2: Boolean = false
```

**10. The if-else expression in Scala**

```
var Var3 =1
```

```
if (Var3 ==1){
 println("True")}else{
 println("False")}
Output: True
```

In the above snippet, the condition evaluates to True and hence True will be printed in the output.

**11. Iteration in Scala**

```
for( a <- 1 to 5){
 println( "Value of a: " + a );
 }
Output:
Value of a: 1
Value of a: 2
Value of a: 3
Value of a: 4
Value of a: 5
```

Scala also supports “while” and “do while” loops.

**12. Declare a simple function in Scala and call it by passing value**

In the below example, the function returns an integer value. Let’s define the function “mul2”:

```
def mul2(m: Int): Int = m * 10
Output: mul2: (m: Int)Int
```

Now let’s pass a value 2 into mul2

```
mul2(2)
Output:
res9: Int = 20
```

**13. Few Data Structures in Scala**

- Arrays

- Lists
- Sets
- Tuple
- Maps
- Option

13.1 Arrays in Scala

```
var name = Array("Faizan","Swati","Kavya", "Deepak", "Deepak")
Output:
name: Array[String] = Array(Faizan, Swati, Kavya, Deepak, Deepak)
```

13.2 List in Scala

```
scala> val numbers = List(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
numbers: List[Int] = List(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

**14. Writing & Running a program in Scala using an editor**

Let us start with a "Hello World!" program. It is a good simple way to understand how to write, compile and run codes in Scala. No prizes for telling the outcome of this code!

```
object HelloWorld {
 def main(args: Array[String]) {
 println("Hello, world!")
 }
```

**15. Advantages of using Scala for Apache Spark**

If you are working with Apache Spark then you would know that it has 4 different APIs support for different languages: **Scala, Java, Python and R**.

Each of these languages have their own unique advantages. But using Scala is more advantageous than other languages. These are the following reasons why Scala is taking over big data world.

- Working with Scala is more productive than working with Java
- Scala is faster than Python and R because it is compiled language
- Scala is a functional language

**16. Comparing Scala, Java, Python and R APIs in Apache Spark**

Let's compare 4 major languages which are supported by Apache Spark API.

| Metrics | Scala | Java | Python | R |
|---|---|---|---|---|
| Type | Compiled | Compiled | Interpreted | Interpreted |
| JVM based | Yes | Yes | No | No |
| Verbosity | Less | More | Less | Less |
| Code Length | Less | More | Less | Less |
| Productivity | High | Less | High | High |
| Scalability | High | High | Less | Less |
| OOPS Support | Yes | Yes | Yes | Yes |

**17. Install Apache Spark & some basic concepts about Apache Spark**

To know the basics of Apache Spark and installation, please refer to my first article on Pyspark. I have introduced basic terminologies used in Apache Spark like big data, cluster computing, driver, worker, spark context, In-memory computation, lazy evaluation, DAG, memory hierarchy and Apache Spark architecture in the previous article.

- **Lazy operation:** Operations which do not execute until we require results.
- **Spark Context:** holds a connection with Spark cluster manager.
- **Driver and Worker:** A driver is in charge of the process of running the main() function of an application and creating the SparkContext.
- **In-memory computation:** Keeping the data in RAM instead of Hard Disk for fast processing.

Spark has three data representations viz RDD, Dataframe, Dataset. To use Apache Spark functionality, we must use one of them for data manipulation. Let's discuss each of them briefly:

- **RDD:** RDD (Resilient Distributed Database) is a collection of elements, that can be divided across multiple nodes in a cluster for parallel processing. It is also fault tolerant collection of elements, which means it can automatically recover from failures. RDD is immutable, we can create RDD once but can't change it.
- **Dataset:** It is also a distributed collection of data. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.). As

I have already discussed in my previous articles, dataset API is only available in Scala and Java. It is not available in Python and R.

- **DataFrame:** In Spark, a DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame. It is mostly used for structured data processing. In Scala, a DataFrame is represented by a Dataset of Rows. A DataFrame can be constructed by wide range of arrays for example, existing RDDs, Hive tables, database tables.
- **Transformation:** Transformation refers to the operation applied on a RDD to create new RDD.
- **Action:** Actions refer to an operation which also apply on RDD that perform computation and send the result back to driver.
- **Broadcast:** We can use the Broadcast variable to save the copy of data across all node.
- **Accumulator:** In Accumulator, variables are used for aggregating the information.

**18. Working with RDD in Apache Spark using Scala**

First step to use RDD functionality is to create a RDD.

```
$ ./bin/spark-shell
```

After typing above command you can start programming of Apache Spark in Scala.

18.1 Creating a RDD from existing source

```
val data = Array(1, 2, 3, 4, 5,6,7,8,9,10)
val distData = sc.parallelize(data)
```

To see the content of any RDD we can use “collect” method. Let’s see the content of distData.

```
scala> distData.collect()
Output: res1: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

18.2 Creating a RDD from External sources

You can create a RDD through external sources such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format. So let’s create a RDD from the text file:

The name of the text file is text.txt. and it has only 2 lines given below.

I love solving data mining problems.

I don’t like solving data mining problems.

Let’s create the RDD by loading it.

```
val lines = sc.textFile("text.txt");
```

Now let's see first two lines in it.

```
lines.take(2)
```

The output is received is as below:

```
Output: Array(I love solving data mining problems., I don't like solving data mining problems)
```

18.3 Transformations and Actions on RDD

18.3.1. Map Transformations

A map transformation is useful when we need to transform a RDD by applying a function to each element. So how can we use map transformation on 'rdd' in our case? Let's calculate the length (number of characters) of each line in "text.txt"

```
val Lenght = lines.map(s => s.length)
Length.collect()
```

After applying above map operation, we get the following output:

```
Output: res6: Array[Int] = Array(36, 42, 37, 43)
```

*18.3.2 Count Action*

Let's count the number of lines in RDD "lines".

```
lines.count()
res1: Long = 4
```

The above action on "lines1" will give 4 as the output

18.3.3 Reduce Action

Let's take the sum of total number of characters in text.txt.

```
val totalLength = Length.reduce((a, b) => a + b)

totalLength: Int = 158
```

18.3.4 flatMap transformation and reduceByKey Action

Let's calculate frequency of each word in "text.txt"

```
val counts = lines.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)

counts.collect()

Output:

res6: Array[(String, Int)] = Array((solving,4), (mining,2), (don't,2), (love,2), (problems.,4), (data,4), (science,2), (I,4), (like,2))
```

18.3.5 filter Transformation

Let's filter out the words in "text.txt" whose length is more than 5.

```
val lg5 = lines.flatMap(line => line.split(" ")).filter(_.length > 5)

Output:

res7: Array[String] = Array(solving, mining, problems., solving, mining, problems., solving, science, problems., solving, science, problems.)
```

**19. Working with DataFrame in Apache Spark using Scala**

A DataFrame in Apache Spark can be created in multiple ways:

- It can be created using different data formats. For example, by loading the data from JSON, CSV
- Loading data from Existing RDD
- Programmatically specifying schema

Let's create a DataFrame using a csv file and perform some analysis on that. For reading a csv file in

Apache Spark, we need to specify a new library in our Scala shell.

```
$ ./bin/spark-shell --packages com.databricks:spark-csv_2.10:1.3.0
```

Now let's load the csv file into a DataFrame df. You can download the file(train) from this link.

```
val df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true").load("train.csv")
```

19.1 Name of columns

```
df.columns

Output:

res0: Array[String] = Array(User_ID, Product_ID, Gender, Age, Occupation, City_Category,
Stay_In_Current_City_Years, Marital_Status, Product_Category_1, Product_Category_2,
Product_Category_3, Purchase)
```

19.2 Number of observations

```
df.count()

Output:

res1: Long = 550068
```

19.3 Print the columns datatype

```
df.printSchema()

Output:

root
 |-- User_ID: integer (nullable = true)
 |-- Product_ID: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- Occupation: integer (nullable = true)
 |-- City_Category: string (nullable = true)
 |-- Stay_In_Current_City_Years: string (nullable = true)
 |-- Marital_Status: integer (nullable = true)
 |-- Product_Category_1: integer (nullable = true)
 |-- Product_Category_2: integer (nullable = true)
 |-- Product_Category_3: integer (nullable = true)
 |-- Purchase: integer (nullable = true)
```

19.4 Show first n rows

```
df.show(2)

Output:

+-------+----------+------+----+----------+-------------+------------------------+--------------+------------------+---------
---------+------------------+--------+
```

```
|User_ID|Product_ID|Gender|
Age|Occupation|City_Category|Stay_In_Current_City_Years|Marital_Status|Product_Category_1|P
roduct_Category_2|Product_Category_3|Purchase|
+-------+----------+------+----+----------+-------------+------------------------+--------------+------------------+---------
---------+-----------------+--------+
|1000001| P00069042| F|0-17| 10| A| 2| 0| 3| null| null| 8370|
|1000001| P00248942| F|0-17| 10| A| 2| 0| 1| 6| 14| 15200|
+-------+----------+------+----+----------+-------------+------------------------+--------------+------------------+---------
---------+-----------------+--------+
only showing top 2 rows
```

19.5 Subsetting or select columns

```
df.select("Age").show(5)
Output:
+-----+
| Age|
+-----+
| 0-17|
| 0-17|
| 0-17|
| 0-17|
| 55+|
+-----+
only showing top 5 rows
```

19.6 Filter rows

```
df.filter(df("Purchase") >= 10000).select("Purchase").show(5)
+--------+
|Purchase|
+--------+
| 15200|
| 15227|
| 19215|
| 15854|
| 15686|
+--------+
```

```
only showing top 5 rows
```

19.7 Group DataFrame

```
df.groupBy("Age").count().show()
Output:
+-----+------+
| Age| count|
+-----+------+
|51-55| 38501|
|46-50| 45701|
| 0-17| 15102|
|36-45|110013|
|26-35|219587|
| 55+| 21504|
|18-25| 99660|
+-----+------+
```

19.8 Apply SQL queries on DataFrame

```
df.registerTempTable("B_friday")
```

Now you can apply SQL queries on “B_friday” table using sqlContext.sql. Lets select columns “Age” from the “B_friday” using SQL statement.

```
sqlContext.sql("select Age from B_friday").show(5)
+----+
| Age|
+----+
|0-17|
|0-17|
|0-17|
|0-17|
| 55+|
+----+
```

**20. Building a machine learning model**

If you have come this far, you are in for a treat! I'll complete this tutorial by building a machine learning model.

I will use only three dependent features and the independent variable in df1. Let's create a DataFrame df1 which has only 4 columns (3 dependent and 1 target).

```
val df1 = df.select("User_ID","Occupation","Marital_Status","Purchase")
```

In above DataFrame df1 "User_ID","Occupation" and "Marital_Status" are features and "Purchase" is target column. Let's try to create a formula for Machine learning model like we do in R. First, we need to import RFormula.

```
import org.apache.spark.ml.feature.RFormula
val formula = new RFormula().setFormula("Purchase ~ User_ID+Occupation+Marital_Status").setFeaturesCol("features").setLabelCol("label")
```

After creating the formula, we need to fit this formula on df1 and transform df1 through this formula.

```
val train = formula.fit(df1).transform(df1)
```

20.1 Applying Linear Regression on train

```
import org.apache.spark.ml.regression.LinearRegression
val lr = new LinearRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
val lrModel = lr.fit(train)
```

Let's print the coefficient and intercept for linear regression.

```
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
Output:
Coefficients: [0.015092115630330033,16.12117786898672,-10.520580986444338] Intercept: -5999.754797883323
```

Let's summarize the model over the training set and print out some metrics.

```
val trainingSummary = lrModel.summary
Now, See the residuals for train's first 5 rows.
trainingSummary.residuals.show(5)
```

```
+-------------------+
| residuals|
+-------------------+
| -883.5877032522076|
| 5946.412296747792|
| -7831.587703252208|
| -8196.587703252208|
|-1381.3298625817588|
+-------------------+
only showing top 5 rows
```

Now, let's see RMSE on train.

```
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
Output:
RMSE: 5021.899441991144
```

Let's repeat above procedure for taking the prediction on cross-validation set

```
val train = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").load("train.csv")
```

Now, randomly divide the train in two part train_cv and test_cv

```
val splits = train.randomSplit(Array(0.7, 0.3))
val (train_cv,test_cv) = (splits(0), splits(1))
```

Now, Transform train_cv and test_cv using RFormula.

```
import org.apache.spark.ml.feature.RFormula
val formula = new RFormula().setFormula("Purchase ~
User_ID+Occupation+Marital_Status").setFeaturesCol("features").setLabelCol("label")
val train_cv1 = formula.fit(train_cv).transform(train_cv)
val test_cv1 = formula.fit(train_cv).transform(test_cv)
```

After transforming using RFormula, we can build a machine learning model and take the predictions.

Let's apply Linear Regression on training and testing data.

```
import org.apache.spark.ml.regression.LinearRegression
val lr = new LinearRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
val lrModel = lr.fit(train_cv1)
val train_cv_pred = lrModel.transform(train_cv1)
val test_cv_pred = lrModel.transform(test_cv1)
```

In train_cv_pred and test_cv_pred, you will find a new column for prediction.

**21. Additional Resources**

- In Scala there are some libraries which are specially written for Data Analysis purpose, refer this link.
- If you want to learn Scala programming refer this link.
- For quick introduction to the Spark API refer this link.
- For, Spark Programming Guide: refer this link.

- To learn about Datasets, and DataFrames, Spark SQL.